Chapter Outline

Chapter 3 Review: Ten Questions and a Course Project

Chapter 3 3D Environment Modeling and Simulation

3.1 Simulation Fundamentals and Tools

3.2 UE Scenes, Rendering, and Interaction

3.3 Multimodal Sensor Data Generation

3.4 Distributed Architecture and Interfaces

3.5 Scene and Vehicle Assets

3.6 Examples and Integrated Practice

3.7 Exercises and Course Projects

4 Units in This Section

3.7.1 Concepts and Principles

Textbook: Sec. 3.4

3.7.2 Calculations and Diagnostics

Bandwidth, timing, and field diagnosis

3.7.3 Design and Course Projects

Textbook: Sec. 3.4

3.7.4 Results and Chapter Summary

Acceptance, timing diagnosis, and summary

3.7.1 Concepts and Principles: The Value of 3D Simulation

Q 1: What Does 3D Simulation Provide?

What are the main roles and benefits of 3D simulation in developing intelligent unmanned systems?

- **Controlled conditions:** Fix the scene, initial pose, and camera; vary lighting or obstacles to explain differences in algorithm behavior.
- **Available data:** Geometry and materials produce images; labels and poses support training and evaluation with less manual collection and labeling.
- **Analyzable feedback:** Control changes states and later observations. Goal-reaching rates and collision records measure task performance under the modeled conditions.

Controlled environments generate observations; feedback supports task comparisons.

3.7.1 Concepts and Principles: Assets and Runtime

Q 2: Connect the Stages of 3D Simulation

Outline the main stages of 3D simulation and explain how they depend on each other.

Geometry and Textures	→	**Scenes and Objects**	→	**Observations / Control**
Asset files		Pose and materials		Data and feedback

- **Asset preparation:** Geometry defines shape; textures and materials define surface response. Import and configure them as usable scene objects.
- **Runtime loop:** State updates objects; rendering and sensors generate observations. Algorithms set motion targets; control and dynamics produce the next state.

Assets must become scene objects before state updates can drive a control loop.

3.7.1 Concepts and Principles: UE Capabilities and Costs

Q 3: Explain the Benefits and Costs of UE

What advantages does Unreal Engine offer for intelligent unmanned system simulation?

- **Visual tasks:** Combine meshes, materials, lights, and cameras in one scene to vary appearance while keeping geometry fixed.
- **Interaction:** Actors and components organize objects. Blueprint events link inputs, variables, and behavior; C++ extends logic and interfaces.
- **Computational cost:** Camera resolution, shadows, and object count increase GPU, CPU, and transfer loads. Keep the task within its frame budget.

Choose engine capabilities to meet specific observation and interaction needs.

3.7.1 Concepts and Principles: A Distributed Control Loop

Q 4: Trace Data Through Distributed Simulation

How do RflySim, UE, CopterSim, and the other modules work together? Explain the benefits and limitations.

Autopilot	→	CopterSim	→	3D Engine	→	Algorithm
Control output		Pose state		Sensor observations		Motion targets

- **Feedback path:** CopterSim loads a dynamics library and computes state. UDP poses update the 3D engine; observations reach the algorithm, whose targets return to the autopilot.
- **Benefit and cost:** Separate model and graphics workloads, but account for message delay, loss, and clock offsets when pairing observations with states.

Separate module roles reduce coupling; communication and timing affect the joint result.

3.7.1 Concepts and Principles: Time Alignment in Q 4

Why Align Time Before Pairing Images and Poses?

An image and a pose have both just arrived from different programs. Do they necessarily describe the same time?

- **Check the clocks first:** Do both timestamps use the same time base?
- **Find the sampling time:** An image describes the scene at sampling, rather than at its arrival time.
- **Match the state:** Select or interpolate the pose at the sampling time; arrival order alone is insufficient.

Evaluate module roles by task load and timing consistency as well as appearance.

3.7.1 Concepts and Principles: Rendering and Modalities

Q 5: Relate the Render Pipeline to Sensor Outputs

Explain the main steps of deferred rendering in UE and how multiple outputs support multimodal sensor generation.

Scene and Visibility	→	**Surface Buffers**	→	**Lighting / Postprocessing**
Select visible geometry		Materials, normals, etc.		Compose the color image

- **Deferred rendering:** Store visible-surface normals and material properties in the G-buffer, then use them in the lighting stage to compute color.
- **Modality branches:** Depth describes geometry; segmentation uses labels. Infrared needs radiation or temperature modeling, rather than RGB recoloring.

Surface buffers support shading; each observation modality also needs its own definition.

Chapter Outline

Chapter 3 Review: Ten Questions and a Course Project

Chapter 3 3D Environment Modeling and Simulation

3.1 Simulation Fundamentals and Tools

3.2 UE Scenes, Rendering, and Interaction

3.3 Multimodal Sensor Data Generation

3.4 Distributed Architecture and Interfaces

3.5 Scene and Vehicle Assets

3.6 Examples and Integrated Practice

3.7 Exercises and Course Projects

4 Units in This Section

3.7.1 Concepts and Principles

Textbook: Sec. 3.4

3.7.2 Calculations and Diagnostics

Bandwidth, timing, and field diagnosis

3.7.3 Design and Course Projects

Textbook: Sec. 3.4

3.7.4 Results and Chapter Summary

Acceptance, timing diagnosis, and summary

3.7.2 Calculations and Diagnostics: RGB Bandwidth

Worked Example: Raw RGB Bandwidth

Given: 1280 × 720, RGB at 8 bits/channel, 30 FPS; ignore compression and protocol overhead.

Pixels per frame	×	**Bytes per pixel**	×	**Frames per second**
1280 × 720		3 × 1 B		30

$$1280 \times 720 \times 3 \times 30 = 82{,}944{,}000 \text{ B/s}$$

1 MiB = 1,048,576 bytes: about 79.1 MiB/s or 663.6 Mb/s, before protocol overhead.

Doubling both image dimensions quadruples pixel count and raw bandwidth.

3.7.2 Calculations and Diagnostics: Time Mismatch

Worked Example: Displacement Over 50 ms

Given: Straight motion at 8 m/s; in one time base, the image is sampled 50 ms after the pose.



$$\Delta t = 0.050\ \mathbf{s}$$

$$\Delta x = v \times \Delta t = 8 \times 0.050 = 0.40\ \mathbf{m}$$

Ignore attitude change and exposure duration. Here 0.40 m is displacement caused by the time mismatch.

Motion during a time mismatch can appear to be a spatial error.

3.7.2 Calculations and Diagnostics: Instance IDs

Diagnostic Example: The Wrong Vehicle Was Read

Given: Read instance 1001. Instances 1001 and 1002 both exist; subscription and reception work.

Copt = ue.getCamCoptObj(1, 1002)
Copt.CopterID → 1002

- **Inference:** type_id=1 queries by instance ID. The call explicitly requests 1002 and returns 1002, so the target mapping is wrong.
- **Correction:** Request 1001 and verify the returned ID. Then check pose, units, and time; do not start by flipping coordinate axes.

Chapter Outline

Chapter 3 Review: Ten Questions and a Course Project

Chapter 3 3D Environment Modeling and Simulation

3.1 Simulation Fundamentals and Tools

3.2 UE Scenes, Rendering, and Interaction

3.3 Multimodal Sensor Data Generation

3.4 Distributed Architecture and Interfaces

3.5 Scene and Vehicle Assets

3.6 Examples and Integrated Practice

3.7 Exercises and Course Projects

4 Units in This Section

3.7.1 Concepts and Principles

Textbook: Sec. 3.4

3.7.2 Calculations and Diagnostics

Bandwidth, timing, and field diagnosis

3.7.3 Design and Course Projects

Textbook: Sec. 3.4

3.7.4 Results and Chapter Summary

Acceptance, timing diagnosis, and summary

3.7.3 Design and Course Projects: Visual Avoidance

Q 6: Design a Visual Obstacle-Avoidance Experiment

Design a RflySim workflow for training a visual UAV obstacle-avoidance algorithm, from scene creation to data generation and interface output.

- **Environment and input:** Specify start, goal, obstacles, and camera mounting. Generate images and states; provide labels required by the training method.
- **Data and feedback:** Record images, timestamps, poses, and actions. Motion targets pass through the autopilot and dynamics to generate the next observation.
- **Evaluation:** Reach within 0.5 m of the goal within 60 s without collision. Count timeout or collision as failure; test independent layouts and lighting.

Keep input conditions and task outcomes for both training and evaluation.

3.7.3 Design and Course Projects: Urban Assets

Q 7: Import a City Block for Ground Navigation

Import a Blender city-block model into RflySim3D for unmanned ground vehicle navigation. Explain integration and optimization.

- **Prepare the model:** Define units, axes, and origin in Blender. Apply transforms, export FBX, and include texture dependencies.
- **Prepare the engine asset:** Import meshes and materials, add road and wall collision, and use levels of detail (LOD) for distant buildings.
- **Platform integration:** Cook for the course engine version; deploy the map and dependencies. Check scale on a road of known length, then check passage and collision locations.

Geometry, materials, and collision together make an asset usable for navigation.

3.7.3 Design and Course Projects: Rays and Point Clouds

Q 8: Design a Ray-Based Point-Cloud Sensor

Describe ray-based LiDAR point-cloud generation for cooperative multi-UAV perception and explain how to balance sampling resolution and accuracy against real-time performance.

Emit Rays
Origin and unit direction

→

Detect Hits
Range and validity

→

Transform and Assemble
Frame and sampling time

- Geometry: A hit is $p = o + r \cdot d$, with unit direction d and range r. Mark misses invalid, then transform points into the agreed frame.
- Workload: 16 beams × 1024 directions × 10 Hz = 163840 rays/s per vehicle. More vehicles add work; fewer rays reduce spatial or temporal resolution.

Each hit point has a defined ray and sampling time.

3.7.3 Design and Course Projects: Q 8 Continued

Estimate Ray Count from the Scan Settings

Four vehicles, 16 beams, 0.2° horizontal steps, 10 revolutions/s: how many rays are emitted?

- Each revolution has 360 ÷ 0.2 = 1800 directions, with 16 rays per direction.

$$1800 \times 16 \times 10 \times 4 = 1\,152\,000 \text{ rays/s}$$

- At 100 FPS, an asynchronous result may be read about 10 ms later on the next frame; measure total latency separately.

Reducing angular density, beam count, or scan rate lowers work and makes data sparser.

3.7.3 Design and Course Projects: Multi-Vehicle Interaction

Q 9: Design a Multi-Vehicle Interaction Demo

Design an interactive multi-vehicle scene using interaction logic and Blueprints. Explain how it works.

Button Event	→	Blueprint Variable	→	Observation / Feedback
Select vehicle 2		SelectedID = 2		Focus 2 and display its ID

- **Interaction design:** A button sets SelectedID; a branch finds that object and updates the label. If missing, report it and keep the previous selection.
- **Platform comparison:** B2 focuses vehicle 2, V changes follow view, and S toggles IDs. Camera-view changes do not send motion targets; leaving selection can restore the view.

Input selects an object; events change state; feedback shows the result.

3.7.3 Design and Course Projects: Large-Scene Performance

Q 10: Plan Large-Scene Performance

Plan optimization and interface use for a Cesium globe or city-scale model on the target laptop.

- Budget: 30 FPS allows 33.3 ms/frame. About 95% of samples lie at or below the 95th-percentile time; record camera rate and algorithm time separately.
- **Target the bottleneck:** Reduce shadows for GPU overload, irrelevant Actors/Tick work for CPU overload, or sampling and transfer load for delayed images.
- **Map and evaluation:** Stream geospatial tiles as needed and state online/offline requirements. Compare frame times and task failure rates on one fixed route.

Reduce the bottleneck load while preserving task geometry and observations.

3.7.3 Design and Course Projects: Two-Camera Specification

Two-Camera Capture and Processing-Time Evaluation

- **Inputs:** Grasslands, vehicle 1, compatible SITL; SeqID 0/1, both 640 × 480 color cameras, requested at 30 Hz.
- **Capture records:** Count changed timestamps per camera; repeated polls do not count. Record 100 frames per camera with ID, receive index, shape, dimensions, and monotonic time.
- **Timing:** Subtract perf_counter() readings around processing to obtain T. For each 100-frame stream, 99 divided by first-to-last receive time gives mean reception rate.
- **Acceptance:** All 200 arrays have shape (480,640,3) and matching dimensions; at least 190 have T ≤100 ms. Without sender sequence numbers, report reception gaps only.

New-frame identifiers track captures; a monotonic clock measures processing. Check each requirement separately.

3.7.3 Design and Course Projects: Two-Camera Setup

Confirm Both Streams Before Timing

How can two 640 × 480 color cameras provide reproducible inputs for the course project?

- **Step 1:** Verify map, vehicle, SeqID 0/1, and camera settings; launch as described in the Readme.
- **Step 2:** Request both pixel streams and record dimensions, channel count, and reception time separately.
- **Step 3:** If images are missing, check requests, configuration, and networking. An empty cache is not a measured zero-frame stream.

3.7.3 Design and Course Projects: New Frames and Timing

Repeated Polling Does Not Mean a New Frame

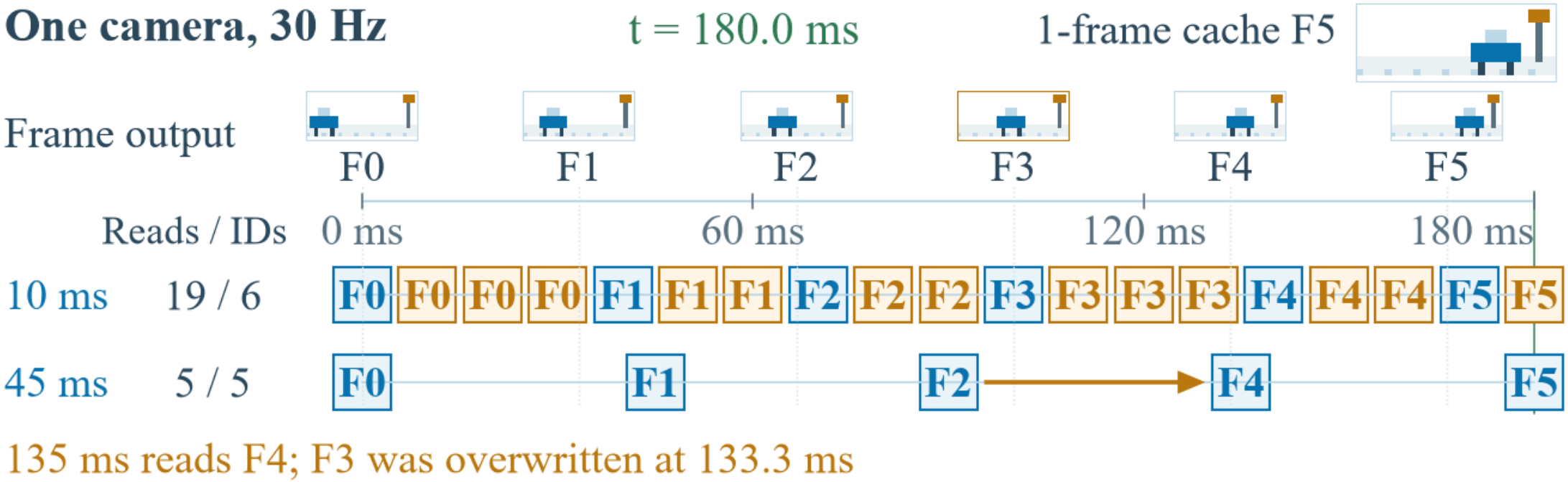
Capture 100 frames per camera. How do you remove duplicate reads and measure per-frame processing?

- Store each previous timestamp; count a new frame only if it changes and valid pixels are available.
- Copy and process that frame. Record elapsed processing time with a monotonic clock.
- Record camera ID, receive index, timestamp, array shape, and processing time.

Deduplicate by timestamp; report processing time separately from the camera period.

3.7.3 Design and Course Projects: Polling One Cache

Polling Rate Controls Duplicate Reads and Overwrites



One camera, two polling schedules, and different observed frame sequences.

Use new-frame identifiers; overwritten cache entries cannot be recovered.

Chapter Outline

Chapter 3 Review: Ten Questions and a Course Project

Chapter 3 3D Environment Modeling and Simulation

3.1 Simulation Fundamentals and Tools

3.2 UE Scenes, Rendering, and Interaction

3.3 Multimodal Sensor Data Generation

3.4 Distributed Architecture and Interfaces

3.5 Scene and Vehicle Assets

3.6 Examples and Integrated Practice

3.7 Exercises and Course Projects

4 Units in This Section

3.7.1 Concepts and Principles

Textbook: Sec. 3.4

3.7.2 Calculations and Diagnostics

Bandwidth, timing, and field diagnosis

3.7.3 Design and Course Projects

Textbook: Sec. 3.4

3.7.4 Results and Chapter Summary

Acceptance, timing diagnosis, and summary

3.7.4 Results and Chapter Summary: Acceptance

Apply the Project Acceptance Criteria

Item	Given result	Decision
Shape of 200 arrays	All (480,640,3)	Shape passes
State width/height	640 / 480	Fields agree
Frames processed in ≤100 ms	188 / 200 frames; need ≥190	Timing fails

Conclusion: Content and dimensions pass; processing time fails. Time saving, conversion, and algorithm execution separately, then retest under the same conditions.

3.7.4 Results and Chapter Summary: Bottleneck Diagnosis

Break a Slow Frame into Measurable Stages

Assumptions: Sequential, non-overlapping stages take 25 ms for conversion, 30 ms for saving, and 65 ms for the algorithm; budget 100 ms.

- Diagnosis: The 65 ms algorithm is largest. Total 120 ms exceeds the budget by 20 ms.

$$25 + 30 + 65 = 120\,\text{ms}; \quad 25 + 30 + 40 = 95\,\text{ms}$$

- Reducing the algorithm to 40 ms gives 95 ms total; remeasure every frame with the same timing boundaries.

Use the same timing boundaries, then retest the slowest stage.

3.7.4 Results and Chapter Summary: Toward Dynamics

From Scene Observations to Vehicle Motion Models

Mass, inertia, forces, and torques determine state changes; the 3D engine uses state to update the image.

Camera configuration and image-array definitions: see teaching unit 3.4.4.

Chapter 4: Equations of Motion, Models, and Simulation Interfaces

https://rflysim.com/doc/zh/2.Course/ch4/index.html

Appearance defines what is seen; dynamics explains how motion develops.

3.7.4 Results and Chapter Summary: Transfer Discussion

Transfer Visual Avoidance from a UAV to a UGV

Reuse the city-block scene in Q 6 for a ground vehicle. Which conditions must be checked again?

- Reuse the scene and the image/timestamp recording workflow; recalibrate the camera mounting.
- Rework vehicle dynamics, tire–ground contact, road boundaries, and collision criteria.
- Verify scale and speed on a road segment of known length, then compare goal-reaching, collision, and timeout outcomes.

Reuse the data chain, then revalidate motion and contact models.

Chapter Summary

Assets, States, Observations, and Evaluation Close the Loop

- **Assets:** Deliver scenes, vehicles, materials, and configuration together, with versions and dependencies.
- **State:** Dynamics or control modules update pose; the 3D engine displays objects by instance ID.
- **Observations:** Cameras, depth sensors, and point clouds have sampling conditions; pair states by timestamp.
- Evaluation: State task metrics, sample counts, and thresholds. Trace failures to inputs, timing, or processing.

Fix inputs and versions to explain differences in simulation results.